User Manual

VIENNA Advantage Silverlight Technical Document

Version 1.0

Table of Contents

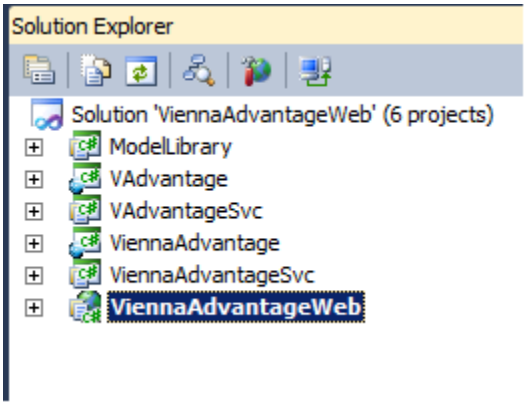
1. INTRODUCTION 3
2. CODE STRUCTURE 4
3. MODULE CREATION 5
3.1 MODULE PREFIX 5
3.2 HOW TO USE PREFIX 6
3.3 RENAME ASSEMBLY 7
3.4 ENTRIES IN MODULE MANAGEMENT WINDOW 8
3.5 LOADING ASSEMBLY 9
4. CONCEPT OF X-CLASS AND M-CLASS 9
4.1 HOW TO MAKE X-CLASS 11
4.2 HOW TO MAKE M-CLASS 13
4.3 PO-CLASS AND DATA FLOW IN CLASSES 16
5. CALLOUT 17
5.1 WRITING A CALLOUT 17
5.2 METHODS USE IN CALLOUT 20
5.3 BIND CALLOUT TO COLUMN 21
5.4 TESTING THE CALLOUT 22
6. PROCESS 23
6.1 WHAT IS PROCESS 23
6.2 EXAMPLE OF A PROCESS CLASS 24
6.3 DEFINING A PROCESS 25
6.3.1 PARAMETER TAB 28
6.4 BINDING OF PROCESS 29
6.5 PROCESS ACCESS 30

1. Introduction

VIENNA Advantage is an open source ERP, developed in dot net technology with visual studio 2010 as IDE (Integrated Development Environment). The Application is in Silverlight, source code of which is available at following link:

https://viennaadvantage.codeplex.com/

After downloaded source code, open it in visual studio 2010. You will see the following structure



There are six projects in the solution.

VAdvantage: It is a Client Side project, which is the part of our base product. The code under this project will run on client side and its dll will reside in Xap file of hosting project.

VAdvantageSvc: It is a Server Side project, which is the part of our base product. The code under this project will run on server side and dll related to this project will reside in bin folder of hosting project. This project contains WCF Service related classes and reference of core project Model Library dll.

ModelLibrary: It is a Core Server Side library, which is the part of our base product. Which contain all the business logic Model classes, Callout classes, Process classes etc. Each class is reside in proper structure inside the ModelLibrary Project. Now the dll of this library is run on server side so this dll is refer in product server side project VAdvantageSvc.

All the above three projects are related to core product.

Further there are two more projects

ViennaAdvantage and VennaAdvantageSvc

The basic concept behind providing these two project is to make your own Module base on the core product.

ViennaAdvantage: It is a Module Client Side project for Module Creation, the code under this project run on client side and it contains the reference of core product VAdvantage project dll.

ViennnAdvatangeSvc: It is a Module Server side project, the code under this project run on the server side and it also contains the reference of core product Model Library dll.

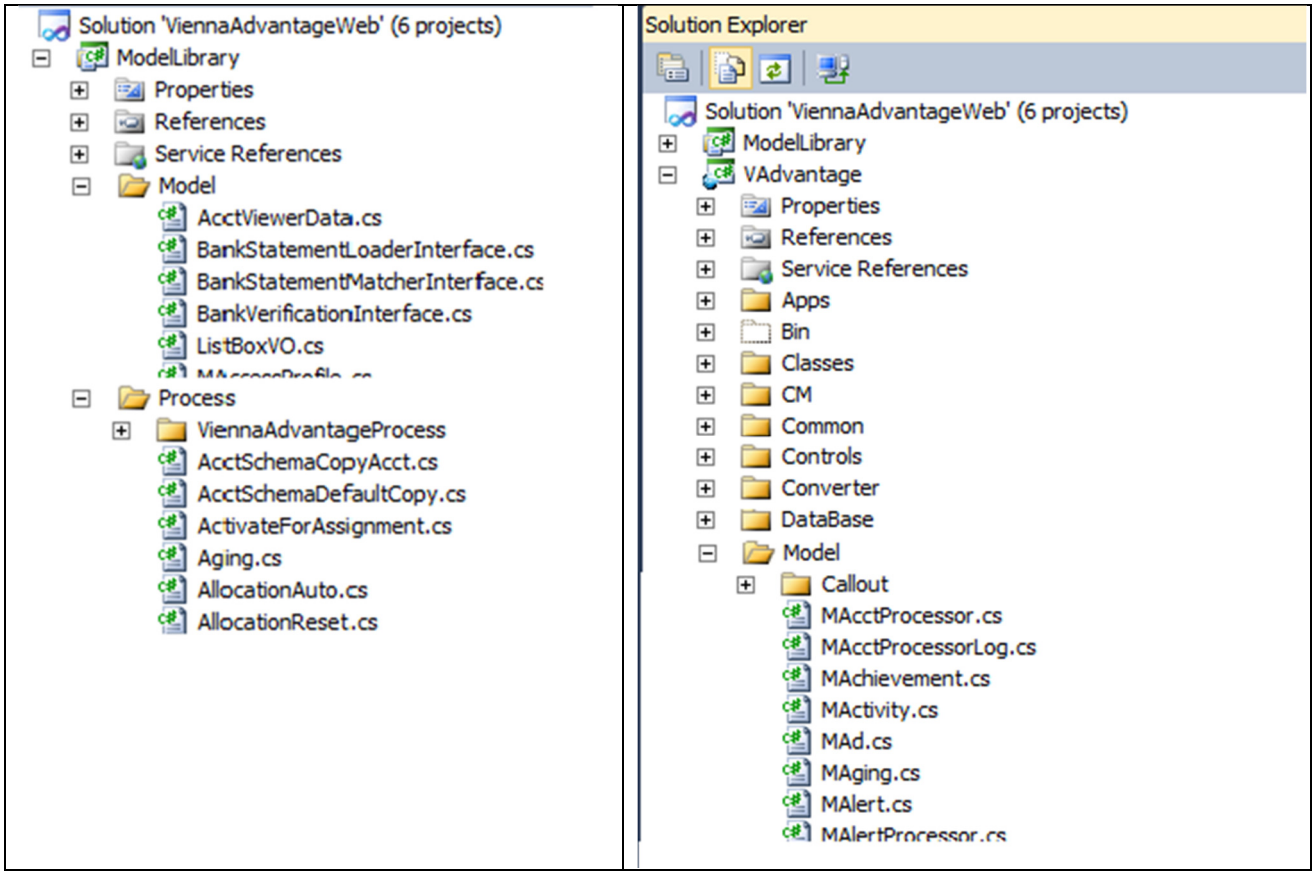
One hosting project named ViennaAdvantageWeb

ViennaAdvatnageWeb: It is the hosting project that finally deploy for hosting in the IIS, which contain reference of all client side Xap and server side dlls.

2. Code Structure

Code structure of Product Project

Model Library Project **VAdvantage Project**



Model Folder: Model Folder contains all business logic Module classes (MClasses) related to Application Dictionary Database Table. If developer doesn't create MClass for a particular database table then data is save directly into database via PO class.

Model Folder will exist in both projects, server side project and client side project.

Model/Callout Folder: Contains all callout classes, which will used for custom logic like on selection of one field populates other fields on the current window tab. Also custom logic for validating user input on the field.

Process Folder: Process Folder contains all process logic classes such as consolidate a number of orders to a single shipment document, copying document, generate invoice on reference button click or on click menu item. Process folder also contains the code for running the third party tool process such as running report. Process will run on sever side so it will be the part of Model library project.

Working with files

Basically you can work in two ways in the solution. One way is that you can add, modify source code of core product itself. Update existing MClasses, Process classes or callout classes OR you can add your MClasses, Process class in the related folder according to your requirement.

Second way is that you work in module specific project in ViennaAdvantage and ViennaAdvantageSvc.

Create your own MClasses, XClasses, Callout classes and put in module specific project and keep the same structure as provided in the product projects. Like keep your MClasses and XClasses in Model Folder on client side and server side. Keep your process classes in Process Folder in ViennaAdvatnageSvc, because process runs on server side.

The benefit of module over directly work in product files is that latter on you can download Module from the Vienna Market. Second, if you directly update product files and host the application with your custom update product dll and later on if you update Vienna Advantage from the Vienna Market then it will replace the product specific dll and your extended work will be lost. But if you work in your specific module then dll related to your module will never update. It will only update if you download your specific module.

3. Module Creation

3.1 Module Prefix

Prefix is the base for any Module in Vienna. At present there are Modules available in the Market and all with the unique prefixes. To name few of them like VACTWZ_ , VAHRUAE_ etc.

For development of any module first step is to get module prefix name from VIENNA. You can also choose your prefix name and send to us so that Vienna will reserve that prefix for your module and will not allocate that name to any other module. Prefix is uniquely define a module separate from other modules.

3.2 How to use Prefix

Whenever you create new database table, column, window etc. use module prefix. Suppose you are going to create Accounting Application and its prefix like ACTAPP. Then use prefix + "_" with every table, column, window etc. E.g if table name is TaxMaster then while create table in Vienna Advantage table name is ACTAPP_TaxMaster.

Note: a) Module prefix name should not be less than three character.

b) Prefix should be unique for every module.

Now after finalize the prefix name, it is mandatory that you use this prefix name in table, column and window in Application Dictionary.

Here is a list of Tables and respective windows where prefix is require:

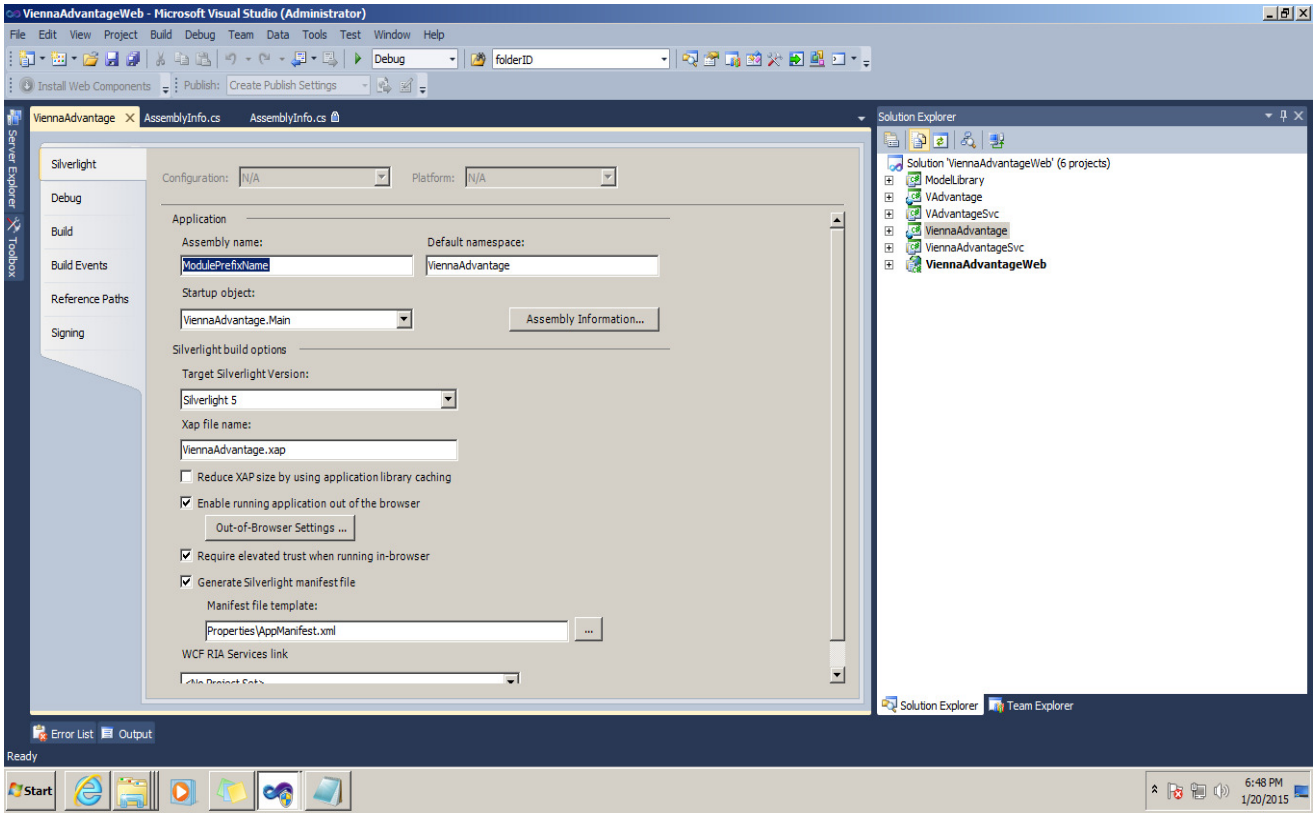
Window Name	Table name	Column	Value	Descripion
Table and Column	AD_Table	DB Table Name		On creating table use prefix name + "_" for DB Table Name. for example if table name is 'TaxMaster'. Then after use prefix it become ' ACTAPP_TaxMaster'. ACTAPP is prefix unique code
System Element	AD_Element	DB Column Name		On creating system element use prefix name + "_" for DB Column Name like column name is ***Description*** *then it become* ***ACTAPP_Description.***
Table and Column	AD_Column	DB Column Name		On creating column use prefix name + "_" for DB Column Name
Message	AD_Message		Value	On creating message use prefix + "_" for Value field
	AD_Reference	Name		On creating reference use prefix + "_" for name field
Window, Tab & Field	AD_Window	Name		On creating window, use prefix + "_" for window name
Report & Process	AD_Process	Name	Value	Use prefix + "_" for name and value filed in window Report & Process
Form	AD_Form	Name		Use prefix + "_" for Name field in F Form window
Shortcut	AD_Shortcut	Name		Use prefix + "_" for Shortcut name
Print Format	AD_PrintFormat	Name		Use prefix + "_" for name
Report Format	AD_ReportFormat	Name		User prefix + "_" for name
Workflow	AD_Workflow	Name	Value	Use prefix + "_" for Name and Value
Validation Rules	AD_Val_Rule	Name		Use prefix + "_" for Name

3.3 Rename Assembly

Client Side Assembly

To run your module, rename the assembly of client side project (ViennaAdvantage) with the module prefix name. For this right click on the project go to properties and change Assembly Name as show in below pic

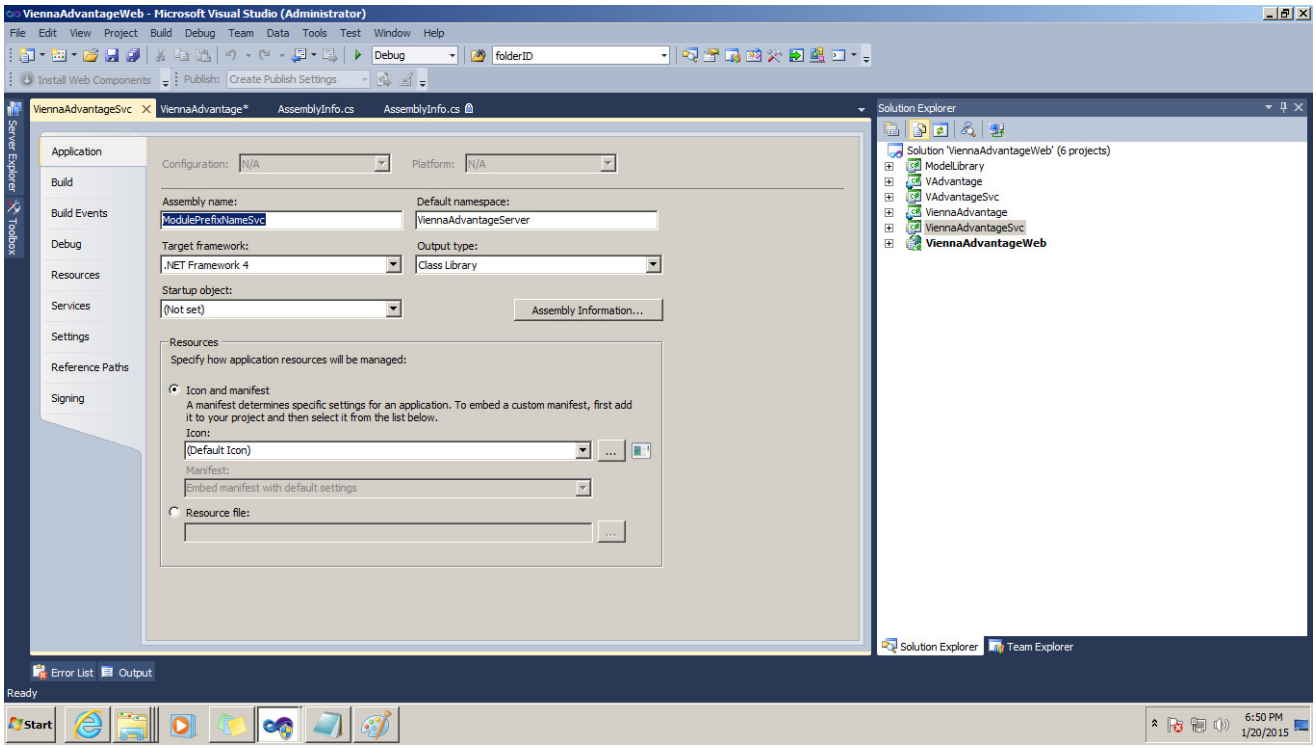
Like if Module prefix name is 'ACTAPP' then Assemble name should be ACTAPP.



Server Side Assembly

Rename Assembly of Server Side Project 'ViennaAdvantageSvc' with Module Prefix Name + "Svc". As show in below pic

So in this case Assemble name is 'ACTAPPSvc'.



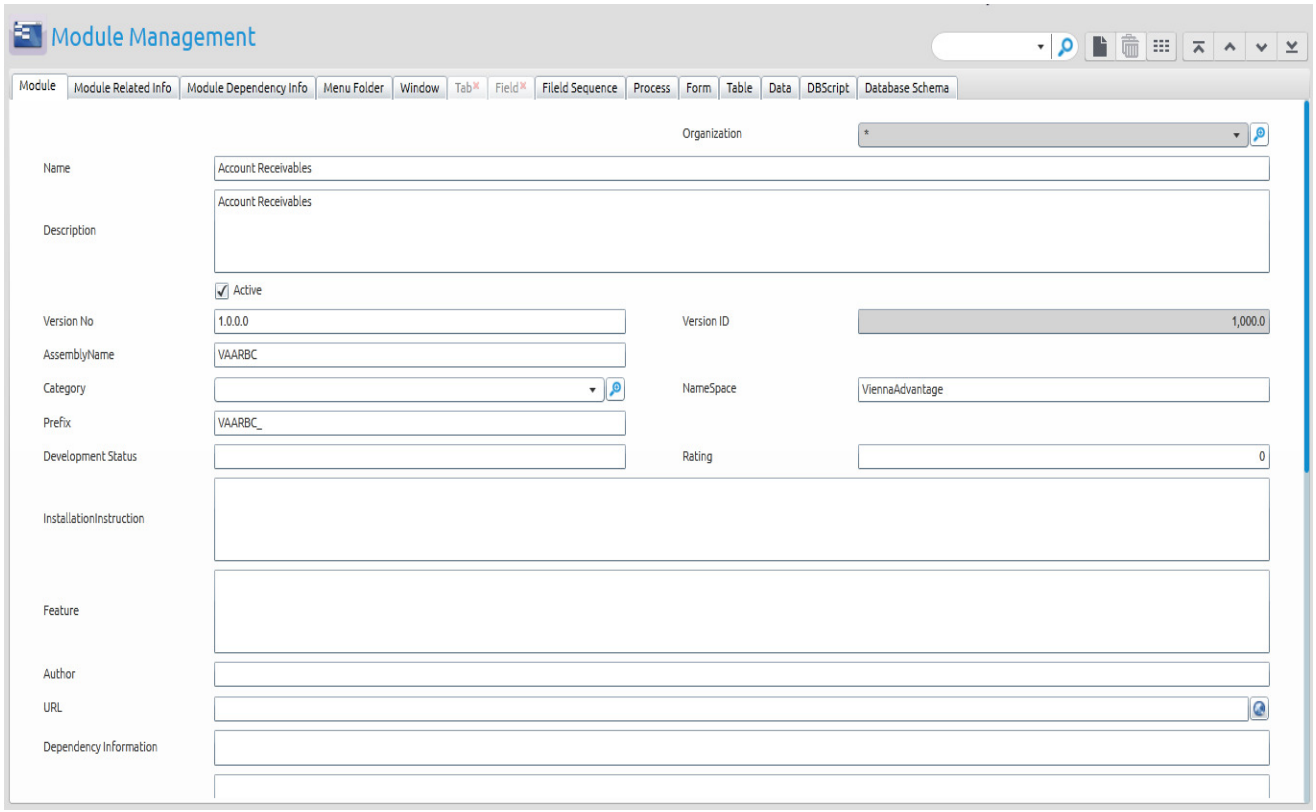
3.4 Entries in Module Management Window

To run your module in the system you must insert module information in Module Management window. After login in the application with Role ‘System Administration’.

Open the window-> Module Management

First Tab of this Window is regarding Module related info

Field Name	Description
Module	Enter your module name
Description	Enter your module description
Version No	Enter your module version number
Assembly Name	Enter your assembly name (Assemble should be created with the same name of prefix),
Unique Prefix	Enter your prefix name + “_” (only one underscore after prefix name)
Namespace	Enter namespace by default it is ViennaAdvantage. But if you change it then insert accordingly



All the fields mentioned above are required to identify the module by system at runtime.

3.5 Loading Assembly

System will load module assemblies which are defined in the module management window and will execute M Classes, X Classess, Callout Classes etc according to assembly.

4. Concept of X-Class and M-Class

Purpose of X Classes is to provide getter and setter method to Model classes. X Classes are the properties classes, which are based on a database tables. Whenever you change the fields of a table you need to update its X class.

M Classes are used to write business logic and it extends X Classes for getter and setter method. If a tab of window using a workflow processes with node have Document Action functionality then M Class should implements DocAction class.

With the help of X-Classes and M Classes, business logic for the window will be created.

Following is an example of an X-Class :

```
namespace ViennaAdvantage.Model
{

/** Generated Model - DO NOT CHANGE */
using System;
using System.Text;
using VAdvantage.DataBase;
using VAdvantage.Common;
using VAdvantage.Classes;
using VAdvantage.Process;
using VAdvantage.Model;
using VAdvantage.Utility;
using System.Data;
/** Generated Model for C_InvoicePaySchedule

 *  @version Vienna Framework 1.1.1 - $Id$ */
publicclassX_C_InvoicePaySchedule : PO
{
public X_C_InvoicePaySchedule (Context ctx, int C_InvoicePaySchedule_ID, String trxName) :
base (ctx, C_InvoicePaySchedule_ID, trxName)
{
/** if (C_InvoicePaySchedule_ID == 0)

{
SetC_InvoicePaySchedule_ID (0);
SetC_Invoice_ID (0);
SetDiscountAmt (0.0);
SetDiscountDate (DateTime.Now);
SetDueAmt (0.0);
SetDueDate (DateTime.Now);
SetIsValid (false);
SetProcessed (false);       // N
}
 */
}
public X_C_InvoicePaySchedule (Ctx ctx, int C_InvoicePaySchedule_ID, String trxName) : base
(ctx, C_InvoicePaySchedule_ID, trxName)
{
/** if (C_InvoicePaySchedule_ID == 0)
{
SetC_InvoicePaySchedule_ID (0);
SetC_Invoice_ID (0);
SetDiscountAmt (0.0);
SetDiscountDate (DateTime.Now);
SetDueAmt (0.0);
SetDueDate (DateTime.Now);
SetIsValid (false);
SetProcessed (false);       // N
}
 */
}
/** Load Constructor
@param ctx context
```

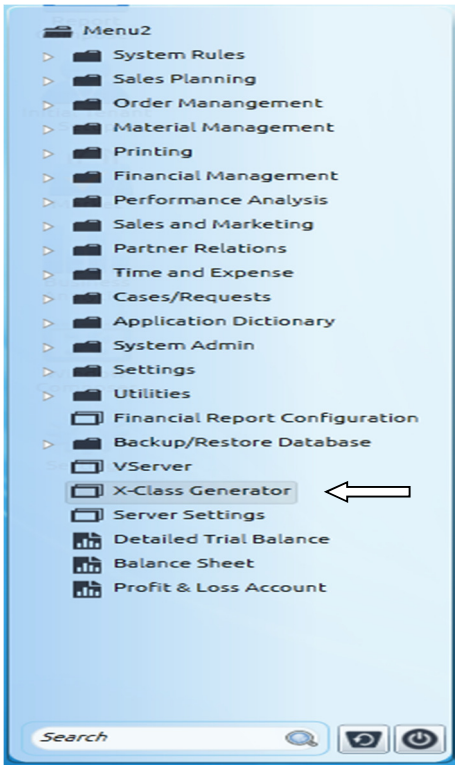
```
@param rs result set
@param trxName transaction
*/
public X_C_InvoicePaySchedule (Context ctx, DataRow rs, String trxName) : base(ctx, rs, trxName)
{
}
/** Load Constructor
@param ctx context
@param rs result set
@param trxName transaction
*/
public X_C_InvoicePaySchedule (Ctx ctx, DataRow rs, String trxName) : base(ctx, rs, trxName)
{
}
/** Load Constructor
@param ctx context
@param rs result set
@param trxName transaction
*/
public X_C_InvoicePaySchedule (Ctx ctx, IDataReader dr, String trxName) : base(ctx, dr, trxName)
{
}
/** Static Constructor
 Set Table ID By Table Name

static X_C_InvoicePaySchedule()
{
 Table_ID = Get_Table_ID(Table_Name);
 model = new KeyNamePair(Table_ID,Table_Name);
}
```

'X-Class' extends the PO class which is a Standard product class and a part of Model Library.

4.1 How to Make X-Class

You can create X-Class by two ways which are

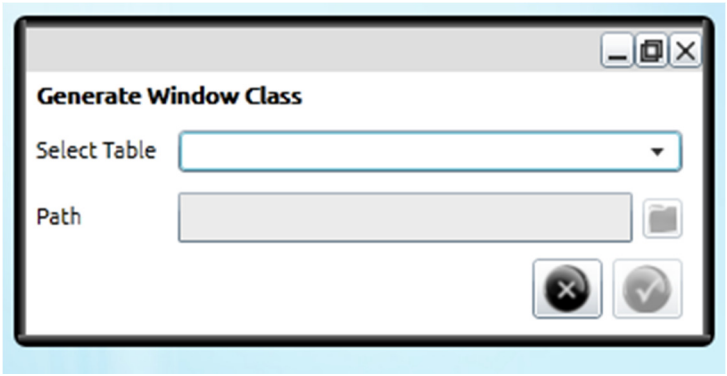
1) With the help of X-Class Generator
2) Manually creations of X-Class



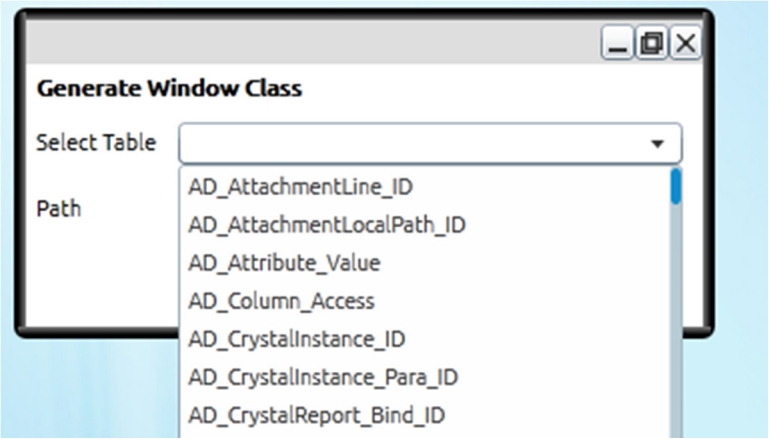
1) With the help of X-Class Generator:

To generate X-Class of a table, you need to install "**Vienna Advantage Professional**" Module from the market after login with role 'System Administrator'. After installation of this module, In Menu you got an "X-Class Generator" link.

- For Creating X-Class, it takes a database connection with Web Config of hosted link.
- Login it with System Administrator.
- After clicking on X-Class Generator from Menu. Pop-Up window will show as below.



- Select the table name and the Path where it can be saved.



- Then press OK.
- X-Class will save on the selected path.

2) Manually X-Class Creation:

You can copy existing X-Class and modify it according to your database table.

Naming Convention of X Class: Use 'X_' + Table Name

For example: Table Name is 'TRN2_Employee'. Its X-Class name should be 'X_TRN2_Employee'. Where X => X-Class symbol in Capitals.

Note: 'TRN2' is a prefix.

It contains all get and set methods of all columns made in a table. It can be on both side projects client side and server side.

If you put X-Class in module specific project then add this class in ViennaAdvantge/Model folder and ViennaAdvantageSvc/Model folder

You can find existing X-Class in Model Folder

4.2 How to Make M-Class

Model Folder contain all the M-Classes. M-class inherit the X-class so that it can access its properties like getter and setter.

Naming conventions for M-Class:

If table name contains prefix with up to 2 characters then ignore prefix and all underscores (_) and append M as prefix to rest of the name. For Example table name is AD_WF_Node then name of M-Class will be MWFNode. If table name is AD_Table then M-Class will be MTable.

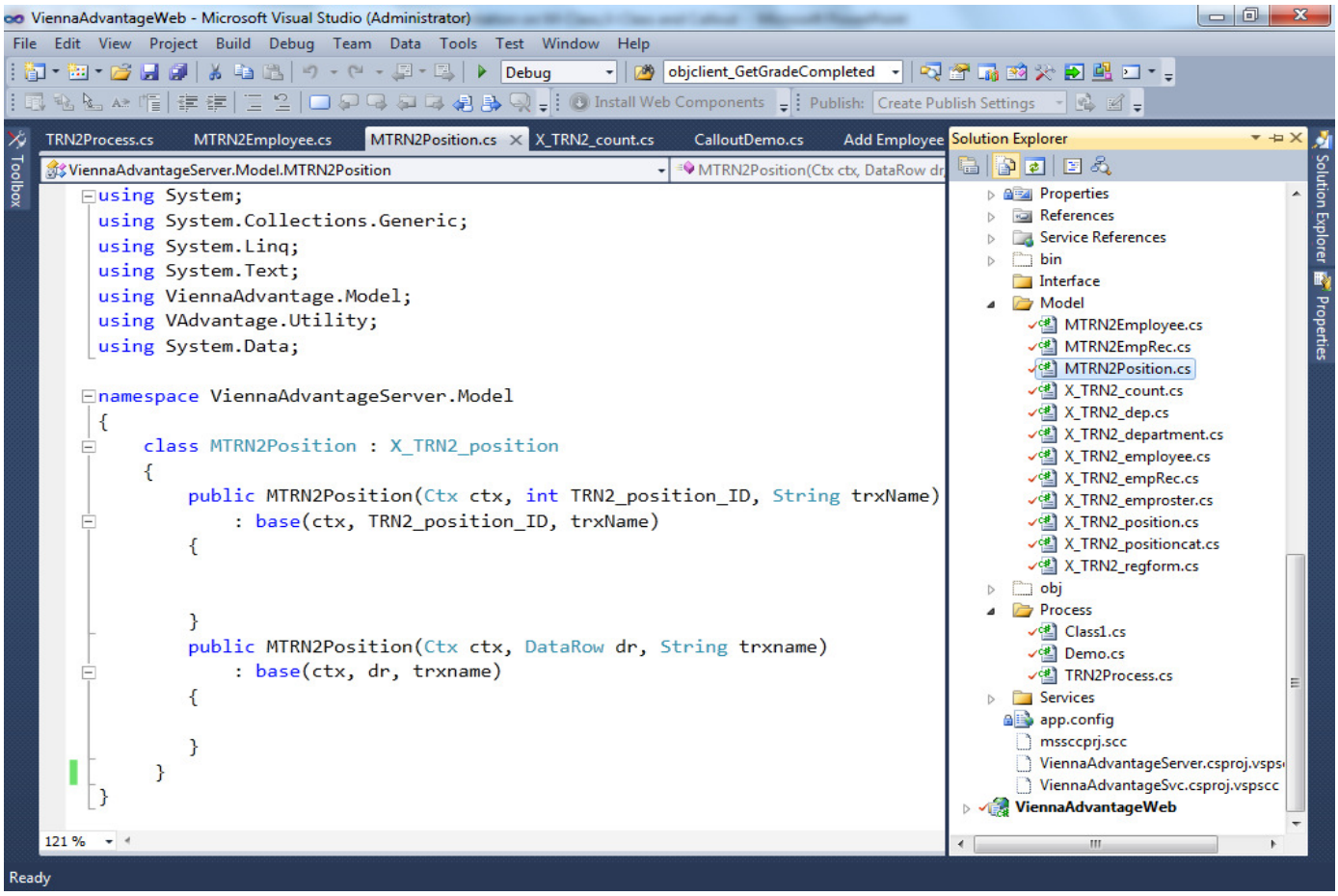
If Table name contains prefix with more than 2 characters then ignore only underscore (_) and append M as prefix to rest of the name. Recommended is to use minimum of three characters in prefix.

For Example table name is WXYZ_WF_Node then name of M-Class will be MWXYZWFNode.

There are four methods on which developer can work. These are:

Before Save:	If user wants to add some business logic before saving the data, then use BeforeSave() method.
Before Delete:	If user wants to add some business logic before deleting the data, then use BeforeDelete() method.
After Save:	If user wants to add some business logic after saving the data, then use AfterSave() method.
After Delete:	If user wants to add some business logic after deleting the data, then use AfterDelete() method.

It has two default parameterize constructors as show in below pic.



1. public MClass (Ctx ctx, int Record_ID, String trxName)

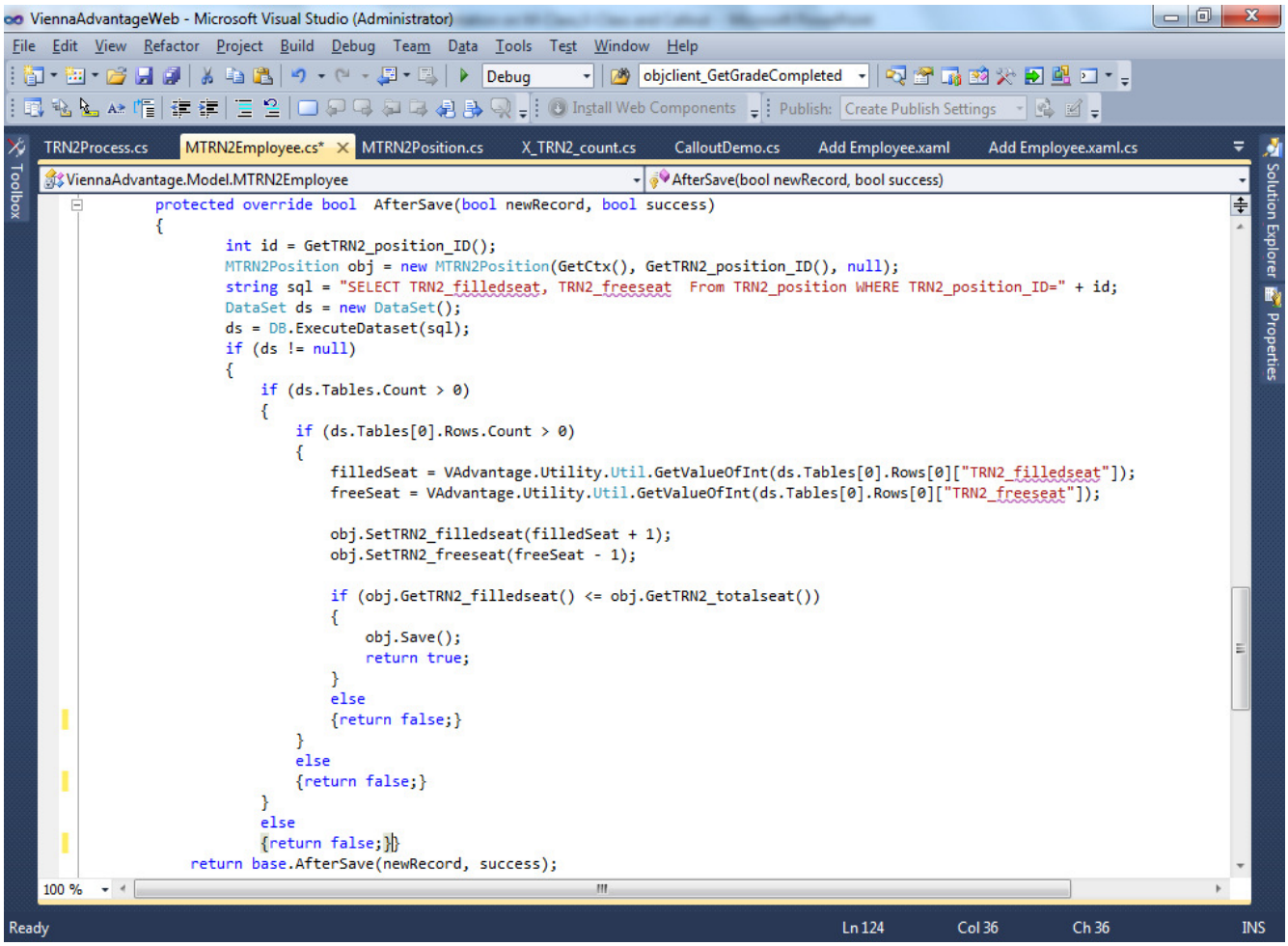
This constructor is used to get a particular record based on Record_ID passed to it and also used to create new record in that table.

If you know a particular Record_ID then this constructor will return that record.

If you pass 0 as Record_ID then it will create a new Record. That record will be saved into database when you call save function with instance of that class.

2. public MClass (Ctx ctx, DataRow rs, String trxName)

When you have Record in your dataset or you have data row for data and you want to initialize MClass for that record then pass that datarow to this constructor. In this way you will reduce time as this will not go to database again to get that data for class instance initialization.



The above example is of Model Folder structure.

For see more M-Classes/X-Classes coding example user can explore below folder

"VAdvantage\Model" Folder, code run at Silverlight client side

"ModelLibrary\Model" Folder, code run at Silverlight server side

If you want to add MClass in Module Projects then you can add class in both side projects Sever side and Client Side project in Model Folder according to your requirement (recommended to add both sides).

4.3 PO-Class and Data Flow in Classes

PO class is persistent object and Base class for actual implementation. It is in ***VAdvantage.Model*** namespace. PO class is the main class, interacts with database for insertion, updation and deletion of the records. PO class is used to set Vienna default value for 7 mandatory fields for each table. All X classes inherits PO class.

Vienna M Class contains business logic, each M Class inherit the corresponding X Class for getter & setter methods and all X Classes inherits PO Class. X Class will communicate with PO Class for saving the data in the database. PO class directly interacts with database to save and retrieve data.

5. Callout

Callout is a coding technique to validate value provided by the user and modify/set the values of other fields in a window. For example: selecting a customer field on window, which then set customer location, contact user and other information's on the window. Callout is always run on client side.

In the following instance, the user if select the employee name from the list then, the position category and position can be determined easily with corresponding Employee ID.

There are mainly two objectives of Callout

1. Validating User Inputs.
2. Setting values to corresponding fields based on User Inputs.

5.1 Writing a Callout

Callout class always extends '**CalloutEngine**' Class. As shown in below example

```
Using System;
using System.Net;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Documents;
using System.Windows.Ink;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Animation;
using System.Windows.Shapes;
using VAdvantage.Model;
using System.Data;
using VAdvantage.Classes;
using VAdvantage.Grids;
using VAdvantage.Utility;
using VAdvantage.Logging;
using VAdvantage.DataBase;

namespace ViennaAdvantage.Model
{
publicclassCalloutInvoice : CalloutEngine
    {
public String Product(Context ctx, int WindowNo, GridTab mTab, GridField mField, Object value)
        {
            if (IsCalloutActive() || value == DBNull.Value || value == null || value.ToString()
== "")
```

```
            {
                return "";
            }
            int M_Product_ID = System.Convert.ToInt32(value);
            if (M_Product_ID == null || M_Product_ID == 0)
                return "";
            SetCalloutActive(true);
            try
            {

                mTab.SetValue("C_Charge_ID", null);

                //   Set Attribute
                if (ctx.GetContextAsInt(Envs.WINDOW_INFO, Envs.TAB_INFO, "M_Product_ID") ==
M_Product_ID
                    && ctx.GetContextAsInt(Envs.WINDOW_INFO, Envs.TAB_INFO,
"M_AttributeSetInstance_ID") != 0)
                {
                    mTab.SetValue("M_AttributeSetInstance_ID",
(int)ctx.GetContextAsInt(Envs.WINDOW_INFO, Envs.TAB_INFO, "M_AttributeSetInstance_ID"));
                }
                else
                {
                    mTab.SetValue("M_AttributeSetInstance_ID", null);
                }

                //try
                //{
                //    object[] pqtyAll = VAdvantage.Classes.InfoLines.PQ.ToArray();
                //    for (int x = 0; x < pqtyAll.Length; x++)
                //    {
                //        object f = pqtyAll.GetValue(x);

                //        int AD_Session_ID =
Util.GetValueOfInt(((VAdvantage.Classes.InfoLines)f)._AD_Session_ID);
                //        int winNo =
Util.GetValueOfInt(((VAdvantage.Classes.InfoLines)f)._windowNo);
                //        Dictionary<int, Decimal> ProductQty =
((VAdvantage.Classes.InfoLines)f)._prodQty;

                //        List<int> key = ProductQty.Keys.ToList();
                //        if (AD_Session_ID == Envs.GetCtx().GetAD_Session_ID() && winNo ==
WindowNo && Util.GetValueOfInt(value) == Util.GetValueOfInt(key[0]))
                //        {
                //            Decimal qty =
Util.GetValueOfDecimal(ProductQty[Util.GetValueOfInt(value)]);
                //            mTab.SetValue("QtyEntered", qty);
                //            mTab.SetValue("QtyInvoiced", qty);
                //            VAdvantage.Classes.InfoLines.PQ.RemoveAt(x);
                //            break;
                //        }

                //    }
                //}
                //catch
```

```
                //{
                //    for (int k = 0; k < VAdvantage.Classes.InfoLines.PQ.Count; k++)
                //    {
                //        VAdvantage.Classes.InfoLines.PQ.RemoveAt(k);
                //    }
                //}

                /*****      Price Calculation see also qty     ****/
                bool isSOTrx = ctx.GetContext(WindowNo, "IsSOTrx").Equals("Y");
                int C_BPartner_ID = ctx.GetContextAsInt(WindowNo, WindowNo, "C_BPartner_ID");
                Decimal Qty = System.Convert.ToDecimal(mTab.GetValue("QtyInvoiced"));
                MProductPricing pp = new MProductPricing(ctx.GetAD_Client_ID(),
ctx.GetAD_Org_ID(),
                        M_Product_ID, C_BPartner_ID, Qty, isSOTrx);

                int M_PriceList_ID = ctx.GetContextAsInt(WindowNo, "M_PriceList_ID");
                pp.SetM_PriceList_ID(M_PriceList_ID);
                int M_PriceList_Version_ID = ctx.GetContextAsInt(WindowNo,
"M_PriceList_Version_ID");
                pp.SetM_PriceList_Version_ID(M_PriceList_Version_ID);
                //long time = ctx.GetContextAsTime(WindowNo, "DateInvoiced");
                //pp.SetPriceDate(time);
                DateTime time = System.Convert.ToDateTime(ctx.GetContext(WindowNo,
"DateInvoiced", false));
                pp.SetPriceDate1(time);
                //
                mTab.SetValue("PriceList", pp.GetPriceList());
                mTab.SetValue("PriceLimit", pp.GetPriceLimit());
                mTab.SetValue("PriceActual", pp.GetPriceStd());
                mTab.SetValue("PriceEntered", pp.GetPriceStd());
                mTab.SetValue("C_Currency_ID", pp.GetC_Currency_ID());
                mTab.SetValue("C_UOM_ID", pp.GetC_UOM_ID());
                ctx.SetContext(WindowNo, "EnforcePriceLimit", pp.IsEnforcePriceLimit() ? "Y" :
"N");
                ctx.SetContext(WindowNo, "DiscountSchema", pp.IsDiscountSchema() ? "Y" : "N");
            }
            catch (Exception ex)
            {
                log.Severe(ex.ToString());
            }
            SetCalloutActive(false);
            return Tax(ctx, WindowNo, mTab, mField, value);
        }
}
}
```

Inside a callout class which inherits from **CalloutEngine,** write a callout method. Signature of the Method in which developer write a logic is

```
Public String DocType(Context ctx, int WindowNo, GridTab mTab, GridField mField, Object value)

{
if (value == DBNull.Value || value == null || value.ToString() == "")
            {
return"";
            }
}
```

There are 5 parameters which are passed in a Callout method.

Parameter	Description
Context	Context of the User Session
WindowNo	Current tab Window Number.
GridTab	Object of the Grid Tab logical refer to table
GridField	Object of Grid Field logical refer to table column
Object	Value of the Field

5.2 Methods Use in Callout

Making Callout Active/InActive

On calling any callout method you can set that callout to active and if callout is already active then don't call it again. See below code example

```
//if callout is active and receiving value parameter to callout is not valid then return

if (IsCalloutActive() || value == DBNull.Value || value == null || value.ToString() == "")

            {

                return "";

            }

//make callout active

SetCalloutActive(true);
```

In the end of the callout method after successfully execute your logic, before finally return from the callout make this callout inactive by below method

```
SetCalloutActive(false);
```

Get/Set Value in Callout

You can get the current active tab fields value with method mTab.GelValue("ColumnName")

```
QtyEntered = Util. GetValueOfDecimal (mTab.GetValue("QtyEntered"));
QtyInvoiced = Util. GetValueOfDecimal (mTab.GetValue("QtyInvoiced"));
```

Following example demonstrates how you can set the values of the fields in the current active tab with method mTab.SetValue("ColumnName",value);

```
mTab.SetValue("PriceActual", PriceActual);
mTab.SetValue("PriceEntered", PriceEntered);
```

You can explore more example of callouts in following folder.

"VAdvantage\Model\Callout", callout run at client side in Silverlight Application

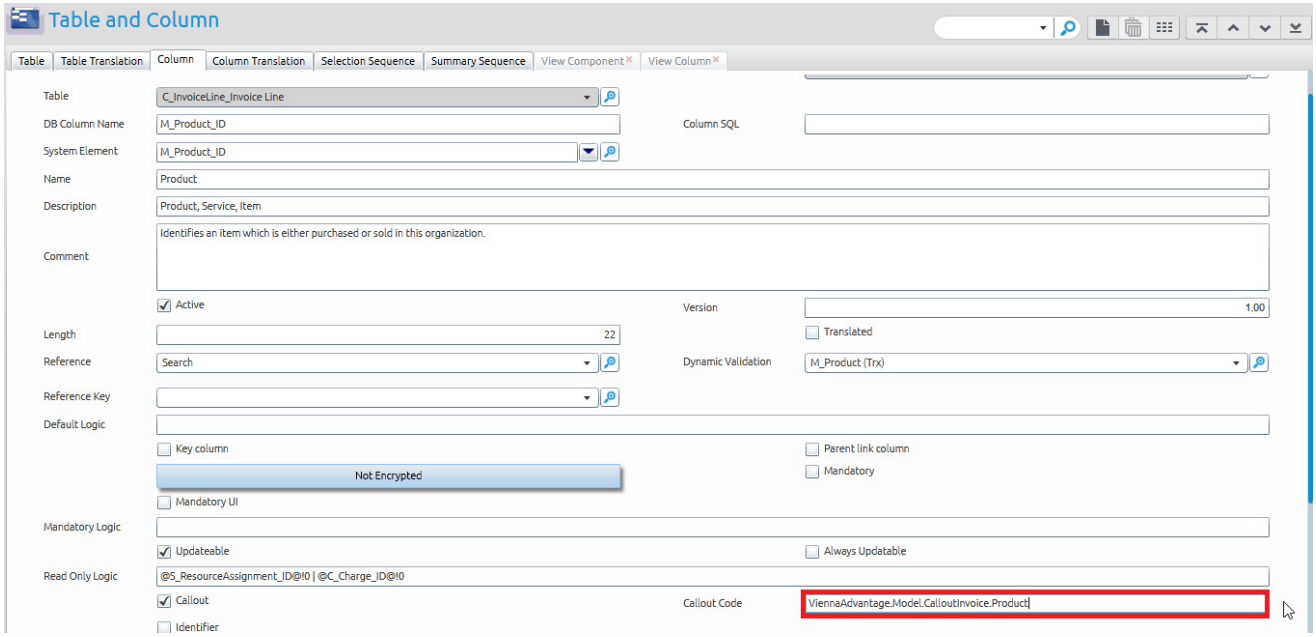
If you want to add callout in module project then add callout class inw folder path

'ViennaAdvantage/Model/Callout' folder

5.3 Bind Callout to Column

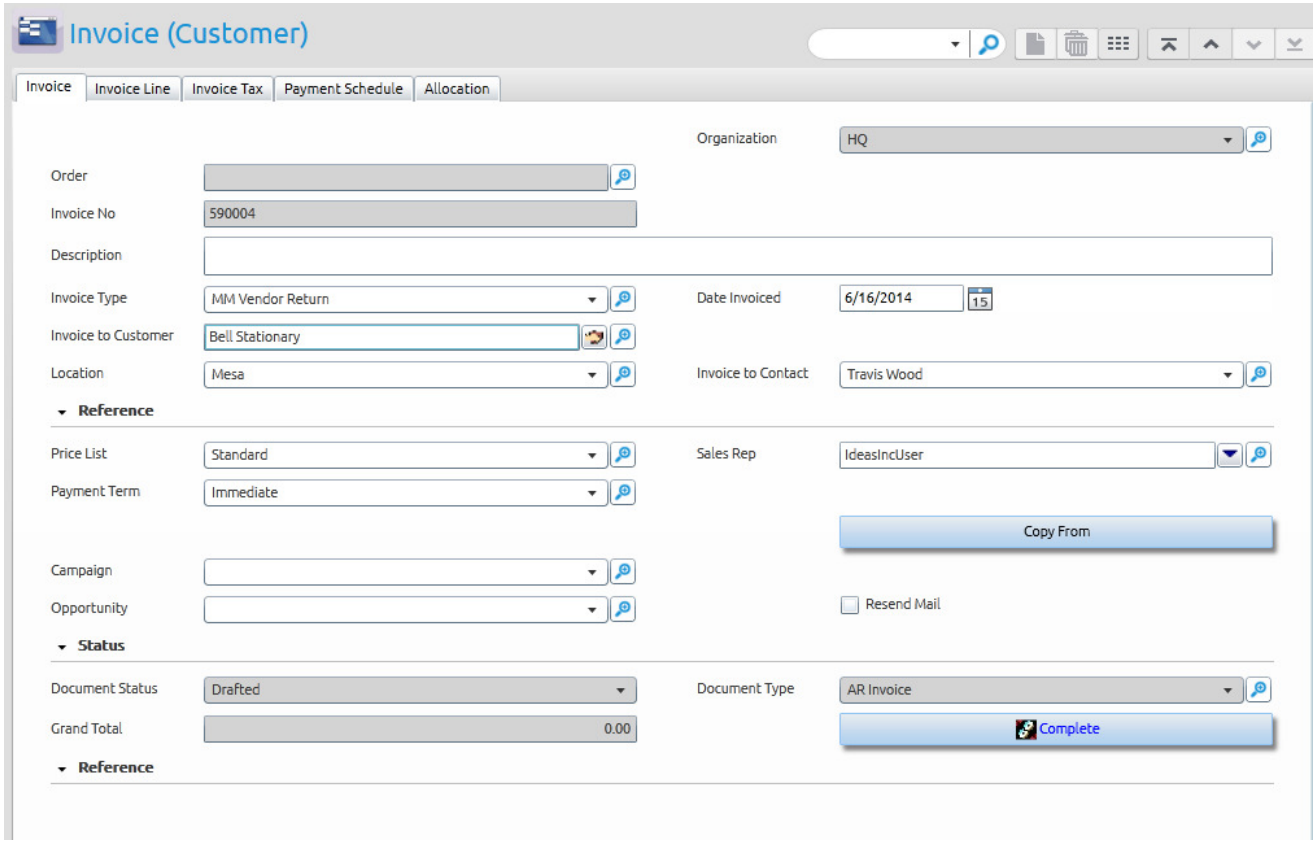
To run callout method, you have to bind a Callout method with a column on third tab ***Column*** of Window named ***Table and Column***. For bind the callout method to a column you have to set **'Callout'** field check box to True and enter the full callout method name with namespace in **Callout Code** textbox.

For example, full name of callout method with namespace of above class is *ViennaAdvantage.Model.CalloutInvoice.Product* enter in the Callout Code textbox as shown in below pic



5.4 Testing the callout

When you enter a value in a field which corresponding column bind with Callout method, it will execute and perform the task what you have coded for. In the given example, when a Customer is selected, it will set other fields like Location, Payment Term etc.



6. Process

6.1 What is Process

Processes can be used to perform some task on a button click or on a menu item click. For example on click you want to consolidate a number of orders to a single shipment document, copying document, generate shipment etc. Process is a class having two methods by default

protected override void Prepare()

protected override String DoIt()'.

It extends the class **'VAdvantage.ProcessEngine.SvrProcess'**

You can also pass parameters which can be retrieved in **'Prepare method'**.

6.2 Example of a Process Class

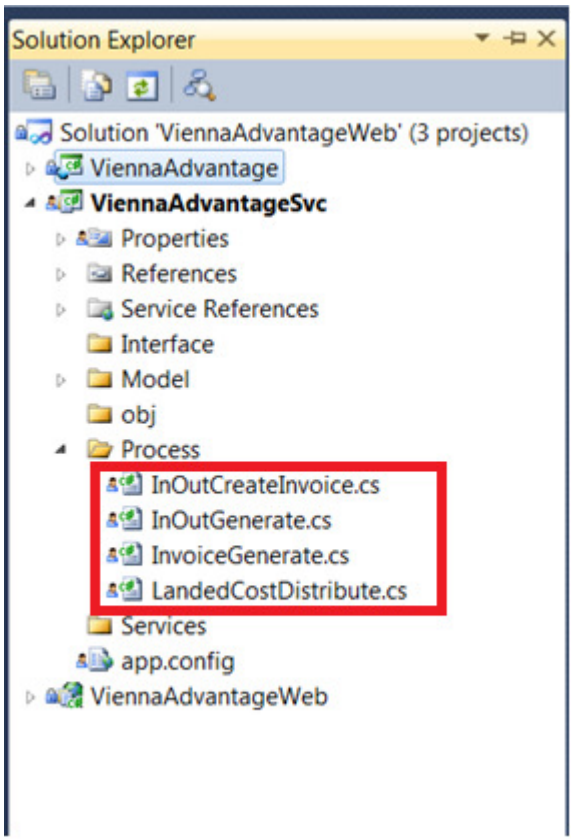
Snippet of a process class is as follows

```
/******************************************************
 * Project Name   : VAdvantage
 * Class Name     : InvoiceGenerate
 * Purpose        : Generate Invoices
 * Class Used     : ProcessEngine.SvrProcess
 * Chronological    Development

  *****************************************************/
using System;
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using VAdvantage.Classes;
using VAdvantage.Common;
using VAdvantage.Process;
using VAdvantage.Model;
using VAdvantage.DataBase;
using VAdvantage.SqlExec;
using VAdvantage.Utility;
using System.Windows.Forms;
using System.Data;
using System.Data.SqlClient;
using VAdvantage.Logging;
using VAdvantage.ProcessEngine;

namespace ViennaAdvantage.Process
{
    public class InvoiceGenerate : VAdvantage.ProcessEngine.SvrProcess
    {
        #region Private Variable
        /**   Manual Selection           */

        #endregion

        /**
        *  Prepare - e.g., get Parameters.
        */
        protected override void Prepare()
        {
            ProcessInfoParameter[] para = GetParameter();
            for (int i = 0; i < para.Length; i++)
            {
                //Traverse parameter loop
            }

        }

        /**
         *    Generate Invoices
         *    @return info
```

```
         *    @throws Exception
         */
        protected override String DoIt()
        {
            //write here business logic
        }


}
```

Process class run on server side so you can find more process classes in core product, in ModelLibrary project. Folder Path is '***ModelLibrary/Process***'

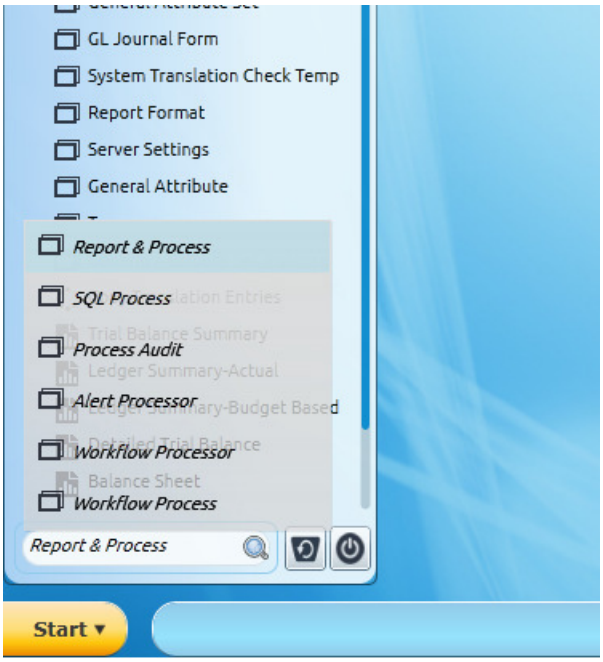
If you want to add Process class in module project then you have to add this class in server side project '**ViennaAdvantageSvc**'. As show in below pic



6.3 Defining a Process

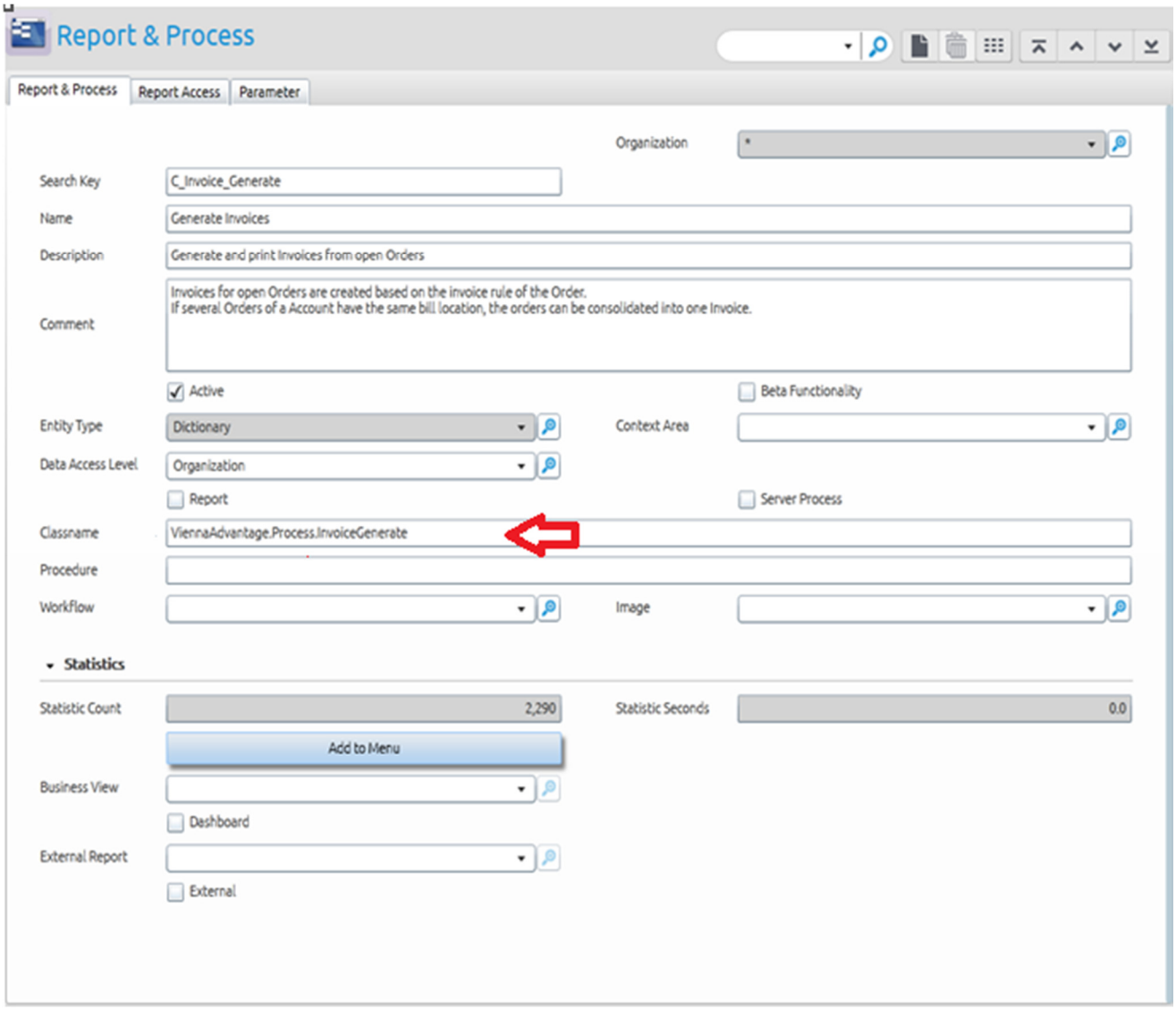
To create a process, you need to enter it in **'Report and Process'** window.

Open Report and Process window after login under System Administrator from menu.



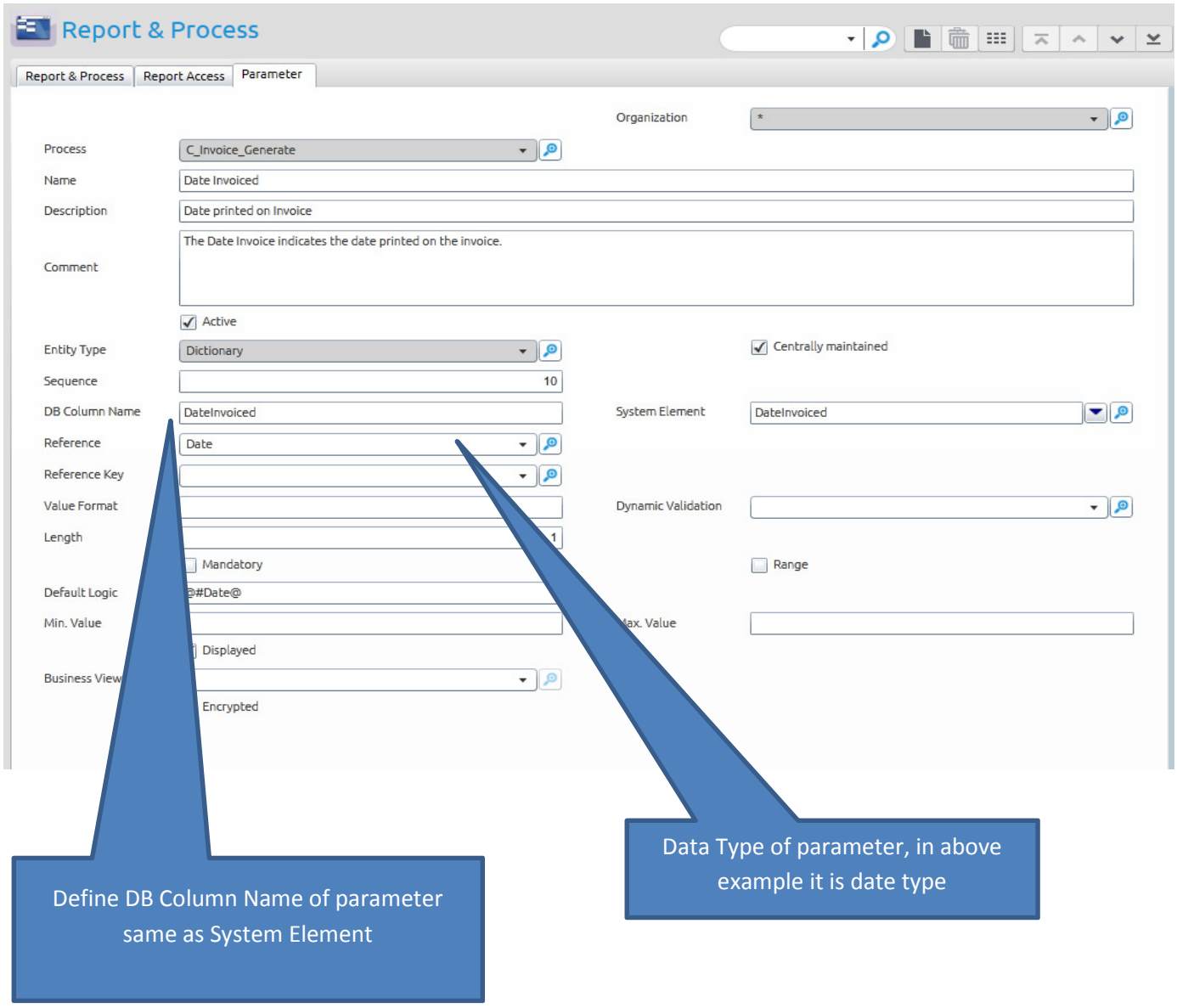
Here, you need to define the proper full name of class (with namespace) of the process to field ***Classname,*** which you are going to use. In the below example, it is

'ViennaAdvantage.Process.InvoiceGenerate'.


Report & Process
Report & Process
Report Access
Parameter
Organization
*
Search Key
C_Invoice_Generate
Name
Generate Invoices
Description
Generate and print Invoices from open Orders
Comment
Invoices for open Orders are created based on the invoice rule of the Order.
If several Orders of a Account have the same bill location, the orders can be consolidated into one Invoice.
Active
Beta Functionality
Entity Type
Dictionary
Context Area
Data Access Level
Organization
Report
Server Process
Classname
ViennaAdvantage.Process.InvoiceGenerate
Procedure
Workflow
Image
Statistics
Statistic Count
2,290
Statistic Seconds
0.0
Add to Menu
Business View
Dashboard
External Report
External

6.3.1 Parameter Tab

If you want to use parameters in process then on parameter tab you can define your parameter

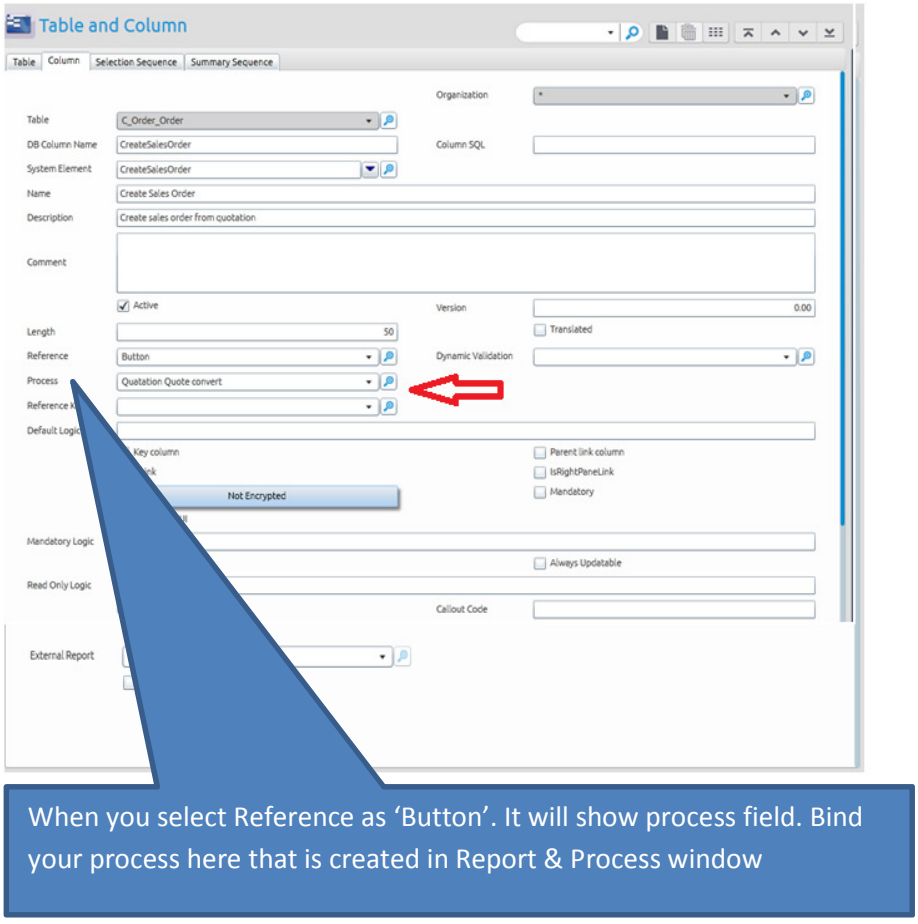


If you define parameter then on click process button or menu item, it will show pop up with parameter that you defined in the parameter tab and it will ask the user for enter parameter value.

6.4 Binding of Process

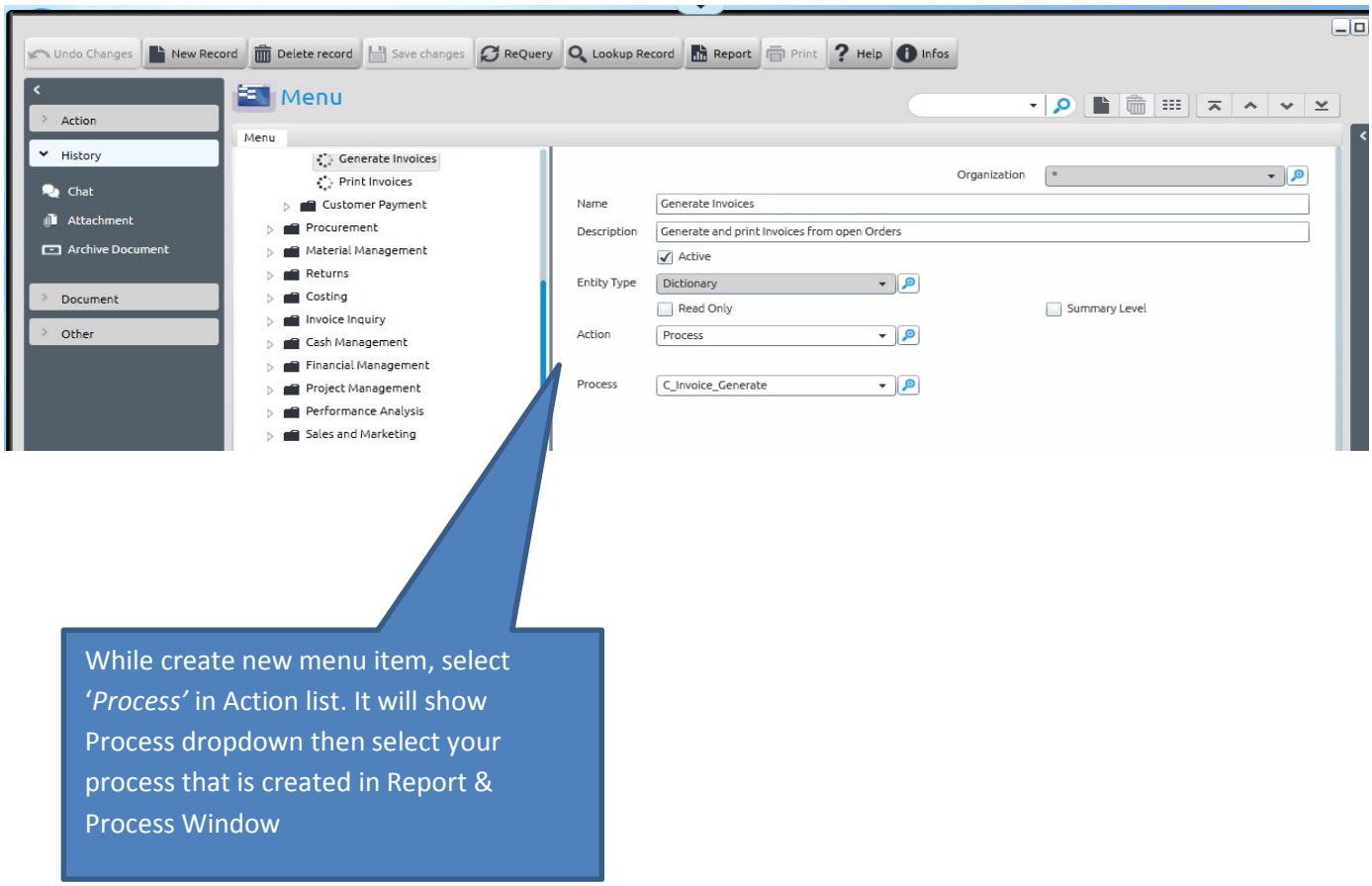
You can bind this process with either button or menu item.

If you want to bind the process with button then this can be defined in table creation level.



When you select Reference as 'Button'. It will show process field. Bind your process here that is created in Report & Process window

And, if you want to run the process on menu item click, then it can be defined in Menu creation level.



6.5 Process Access

You have to give access to Role whose user can access the process.

There are two approaches to achieve it

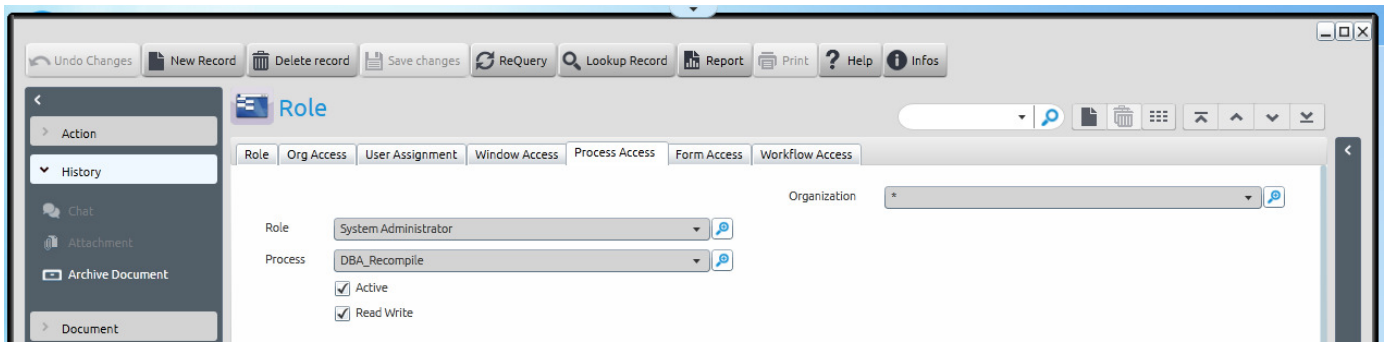
First: Go to Report and Process window-> Go to second tab **Process Access**. Here you can give access to Role

Second: Login in the application with Client Admin Role

Open the Role Window as shown below->

Select the Role to which you give access. For example select '***Client Admin***' Role from the Role Tab (First Tab)

Go to **Process Access** tab



Press New Record button. Select **Process** and set **Active** and **Read Write** checkbox to True

Save the record. Access is assigned to selected role

Re login in the application for see that changes